**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

**Лабораторная работа № 1**

по курсу «Машинное обучение»

Студент: Гаврилов М.С.

Группа: 80-306б

Преподаватель:

Оценка:

Москва, 2022

1. **Постановка задачи**

Формально говоря вам предстоит сделать следующее:

1) реализовать следующие алгоритмы машинного обучения: Linear/ Logistic Regression, SVM, KNN, Naive Bayes в отдельных классах

2) Данные классы должны наследоваться от BaseEstimator и ClassifierMixin, иметь методы fit и predict (подробнее: https://scikit-learn.org/stable/developers/develop.html)

3) Вы должны организовать весь процесс предобработки, обучения и тестирования с помощью Pipeline (подробнее: https://scikit-learn.org/stable/modules/compose.html)

4) Вы должны настроить гиперпараметры моделей с помощью кросс валидации (GridSearchCV,RandomSearchCV, подробнее здесь: https://scikit-learn.org/stable/modules/grid\_search.html), вывести и сохранить эти гиперпараметры в файл, вместе с обученными моделями

5) Проделать аналогично с коробочными решениями

6) Для каждой модели получить оценки метрик:Confusion Matrix, Accuracy, Recall, Precision, ROC\_AUC curve (подробнее: Hands on machine learning with python and scikit learn chapter 3, mlcourse.ai, https://ml-handbook.ru/chapters/model\_evaluation/intro)

7) Проанализировать полученные результаты и сделать выводы о применимости моделей

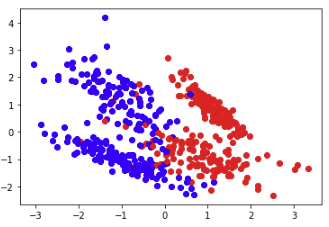
8) Загрузить полученные гиперпараметры модели и обученные модели в формате pickle на гит вместе с jupyter notebook ваших экспериментов

1. **Выполнение работы**
2. Реализация собственных версий алгоритмов машинного обучения.

В этом разделе демонстрируются классы, реализующие модели Линейной/Логистической регрессии, SVM, KNN, Наивный Байесовский алгоритм. Также показана работоспособность моих реализаций требуемых метрик.

Для тестирования алгоритмов используется искусственный датасет для бинарной классификации, созданный с помощью skl.

Вот его визуализация:



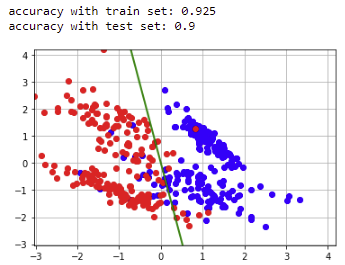
* 1. Линейная / логистическая регрессия

1.1.1. Линейная регрессия

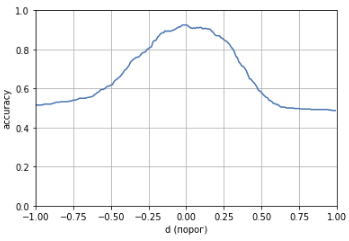


Метрики для данной модели.

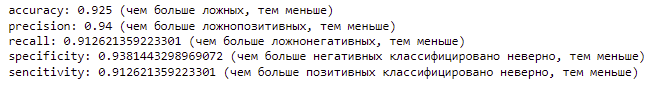
Разделяющая прямая:



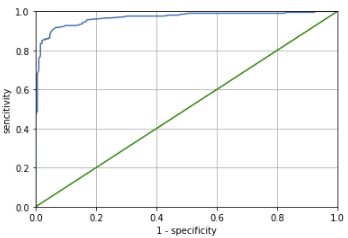
Кривая «порог – точность»:   
Она очень помогает в выборе оптимального значения порога



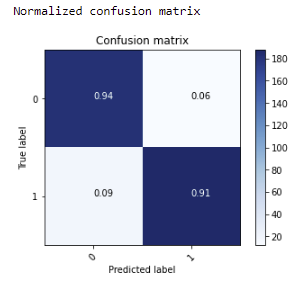
Основные метрики бинарной классификации:



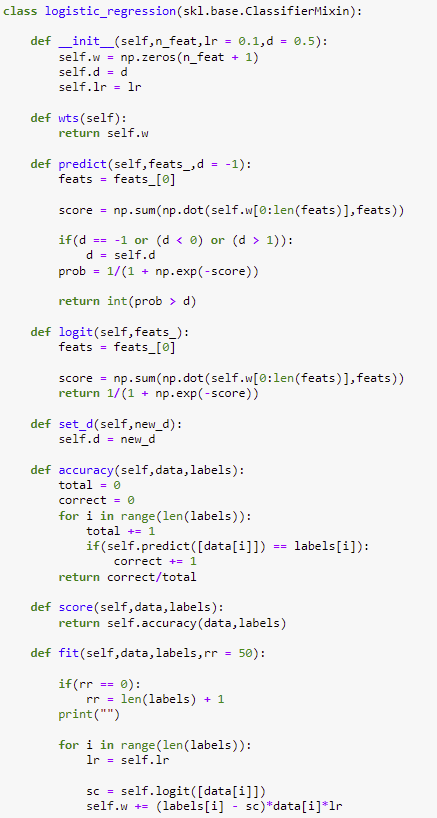
Кривая ROC:



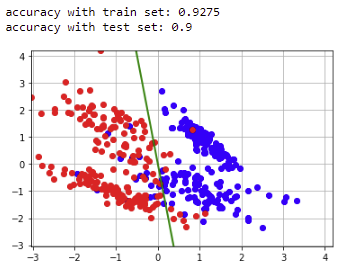
Матрица спутывания:



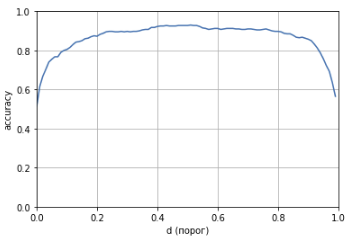
1.1.2. Логистическая регрессия



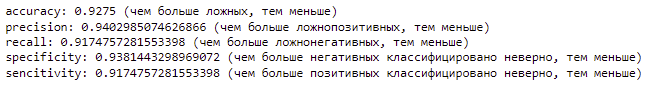
Разделяющая прямая:



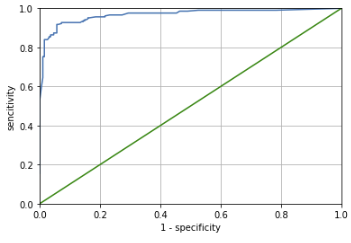
Кривая «порог – точность»:



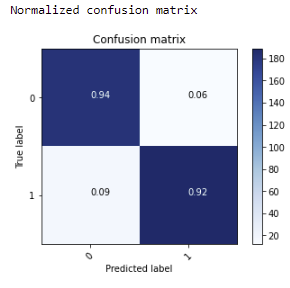
Основные метрики бинарной классификации:



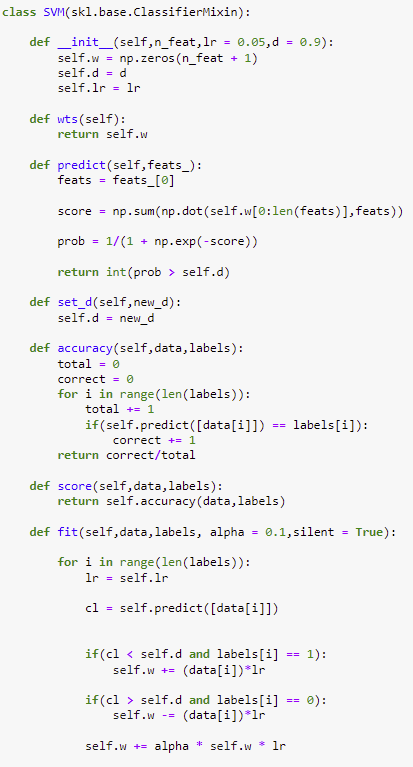
Кривая ROC:



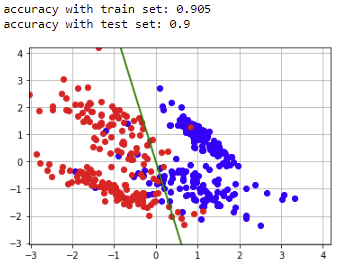
Матрица спутывания:



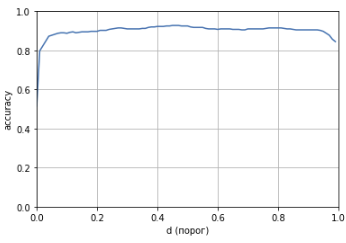
* 1. SVM



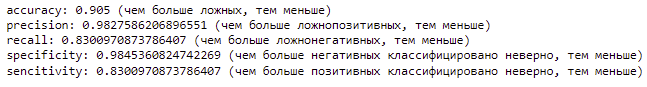
Разделяющая прямая:

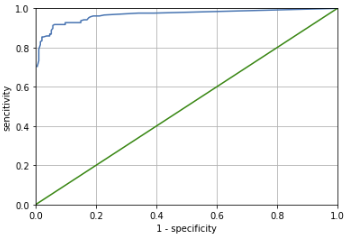


Кривая «порог – точность»:

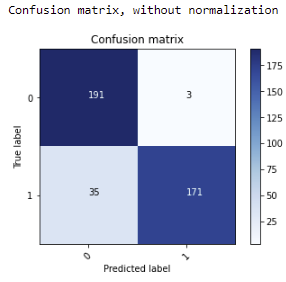


Основные метрики бинарной классификации:

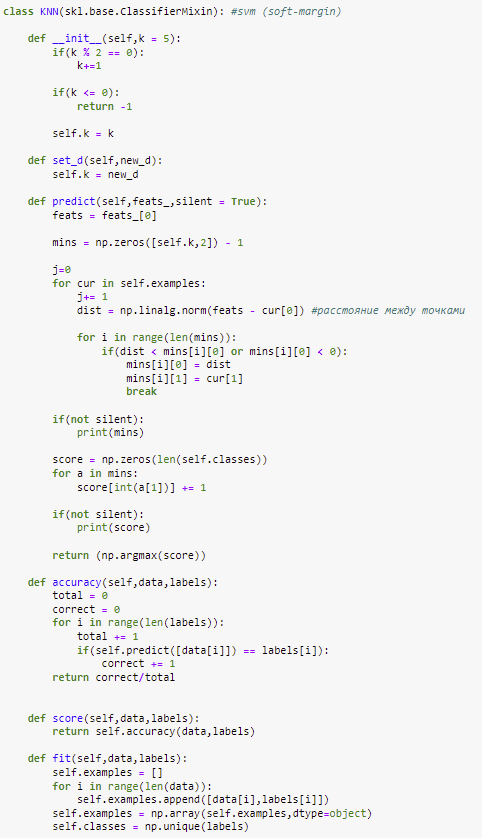




Матрица спутывания:



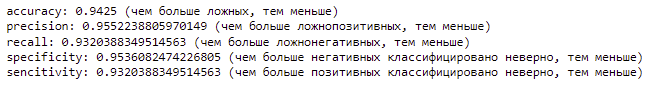
1.3. KNN



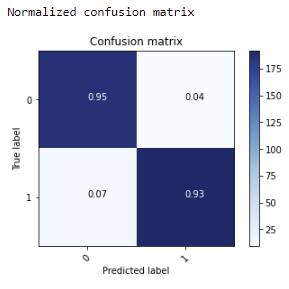
Точность на обучающей и тестовой выборке:



Основные метрики бинарной классификации:



Матрица спутывания:



Так как алгоритм KNN не включает в себя порогового значения или его аналогов, кривые ROC и порог-точность не применимы

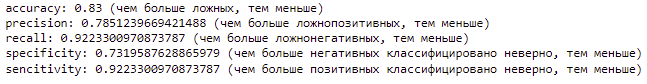
1.4. Наивный Байесовский алгоритм



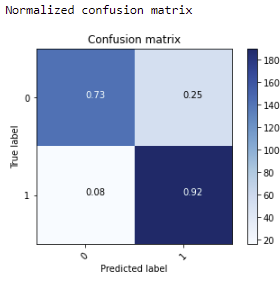
Точность на обучающей и тестовой выборке:



Основные метрики бинарной классификации:



Матрица спутывания:



Интересно, что, не смотря на специфику классических применений этого алгоритма, он довольно универсален. Например, для приемлемой классификации точек на двумерной плоскости, я округлил их координаты и представил координатные пары, как признаки. Можно предположить, что, если увеличить разрешение (округлять не до целых, а до 0.5, например), то точность еще повысится, но это не основная тема лабораторной, так что предположение осталось непроверенным.

Теперь можно проанализировать алгоритмы и сделать некоторые выводы об их особенностях. Во-первых, благодаря кривой порог-точность, можно заметить основную разницу между линейной регрессией, логистической регрессией и методом опорных векторов с точки зрения пользователя: Линейная регрессия имеет лишь небольшой диапазон значений дискриминанта, в котором точность максимальна. При использовании логистической регрессии этот диапазон существенно расширяется, но все равно ближе к краям точность падает. У SVM же, благодаря максимизации зазора, точность примерно равна почти на любых значениях порога, пусть и не всегда столь же высока, как максимальная точность линейной / логистической регрессии.

Самым точным алгоритмом оказался KNN, самым неточным – Наивный Байесовский (хотя его показатели, наверное, можно улучшить, изменив выбор признаков)

Несмотря на, казалось, бы, очевидное превосходство KNN, у него есть существенный недостаток – все классифицируемые объекты должны принадлежать пространству одной размерности. Наивный Байесовский алгоритм же может принимать на вход объекты разной размерности и, при этом, успешно их классифицировать.

1. Реализация поставленной в нулевой лабораторной работе задачи.

Поставленная задача:

Классификация слов русского языка по частям речи.

Так как при классификации слов я, так или иначе, буду подавать на вход алгоритмам буквы алфавита, я сомневаюсь в пригодности для решения моей задачи алгоритма линейной регрессии и ему подобных, так как решение задачи регрессии подразумевает нахождение в непрерывном векторном пространстве, в то время как пространство букв алфавита дискретно, не векторно, и не может быть спроецировано на векторное пространство (то, что буквы «А» и «Б» находятся рядом в алфавите, нахождение буквы «А» на конце слова не делает его похожим на слово с буквой «Б» на конце)

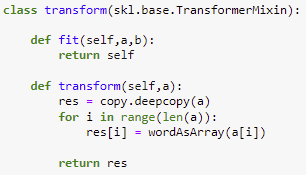
Я предполагаю, что наибольшего успеха получится добиться при использовании наивного Байесовского классификатора, так как он не работает с признаками, как с координатами. Также, возможно, KNN даст неплохой результат благодаря нахождению точных совпадений, и потому что он лучше всех себя показал при тестировании на искусственном датасете, может это о чем-то говорит.

При классификации будет использоваться датасет из слов, относящихся к одной из 7 частей речи: имя существительное, предлог, имя прилагательное, глагол, наречие, союз, причастие.

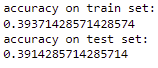
* 1. Наивный подход

Для начала попробуем использовать в качестве признаков буквы слова – каждая буква, входящая в слово, есть отдельный признак. Единственный доступный при таком подходе классификатор – наивный Байесовский.

Используем такую функцию-трансформер (wordAsArray = слово в виде массива алфавитных индексов букв)

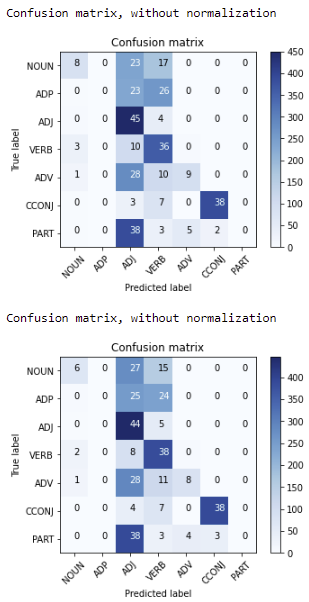


Точность, хоть и лучше случайного угадывания (0.14 для семи классов), но, все же, неприемлемо низкая.



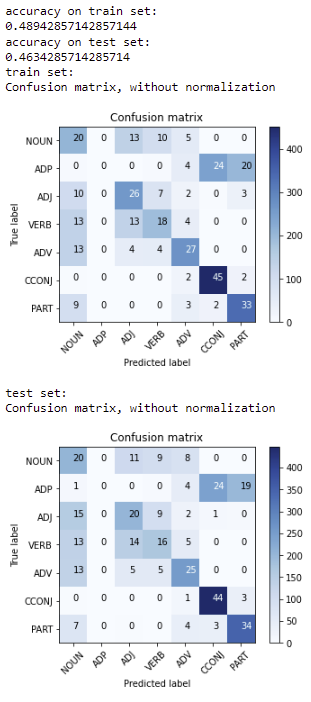
Матрицы спутывания также выглядят неважно:

**Все числа на матрицах спутывания, представленных ниже, уменьшены в 10 раз, чтобы корректно уместить их на рисунке.**



2.2. Выделение важных признаков.

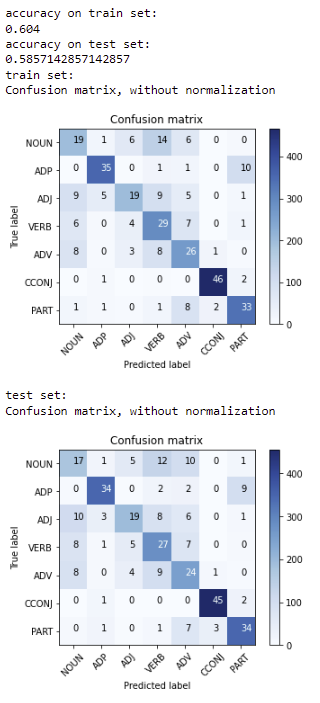
Будем использовать массив из признаков, которые являются ключевыми для определения части речи. Для начала включим в него только длину слова. Используемый классификатор – по-прежнему – наивный Байесовский.



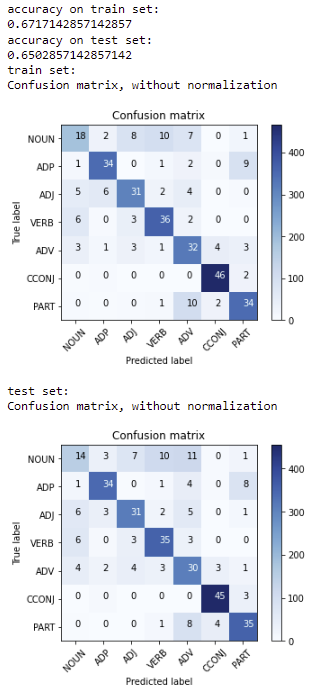
Результат уже лучше. Видно, что, так как предлоги и союзы имеют одинаковую длину, модель совершенно их не различает.

Проверим, что получится, если в массив добавить первую букву слова, последнюю букву слова и различные комбинации трех вышеперечисленных признаков.

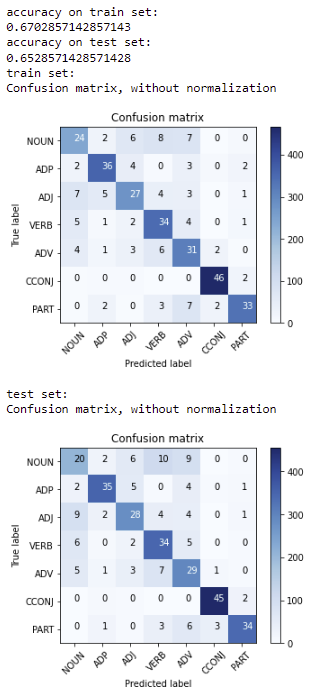
**Длина + первая буква**

****

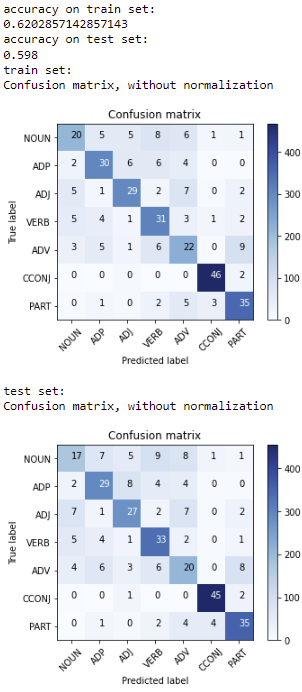
**Длина + последняя буква**

****

**Длина + обе буквы**

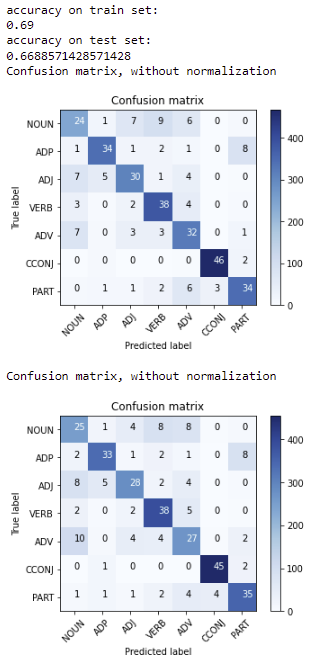


**Обе буквы без длины**



По сути, добавление любой буквы существенно увеличивает точность предсказаний и приводит матрицу спутывания в приличный вид. Ожидаемо, что длина + обе буквы – лучшая комбинация, интересно, что обе буквы без длины тоже показали неплохой результат.

Наилучший результат получается, если скомбинировать длину, первую букву, последнюю букву, предпоследнюю букву и букву перед предпоследней.



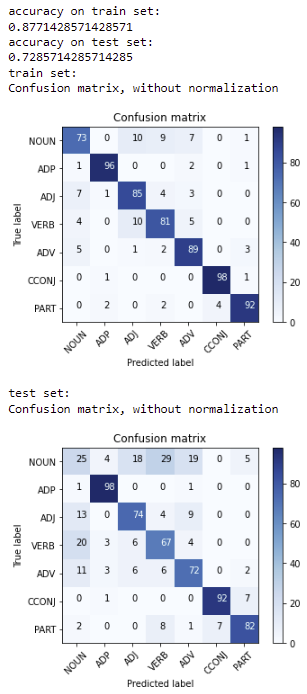
Даже с оптимальным массивом признаков достичь высоких значений точности, используя наивный Байесовский алгоритм не получается. Следующий алгоритм, который будет испытан – KNN.

Используется трансформер, строящий лучший массив признаков:



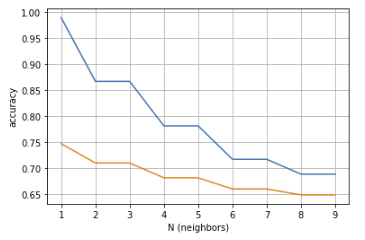
Так как моя реализация KNN никак не оптимизирована и работает крайне медленно, я сгенерировал уменьшенный датасет в 100 экземпляров для тренировки и 100 для тестирования.

Метрики полученной модели: (на матрицах спутывания истинные числа)



Получена поразительная точность. Идея использования KNN определенно стоит дальнейших исследований.

Определим, какое число соседей дает наилучшую точность.



По графику видно, что чем меньше соседей рассматривается, тем лучше. При k = 1 классификатор превращается в своего рода словарь, который ищет классифицируемое слово в памяти, и, если не находит, выдает наиболее близкое по признакам слово.

Почему KNN работает?

KNN, также, как и регрессии основан на представлении данных, как координат в гиперпространстве параметров, потому кажется странным, что, несмотря на то, что буквы алфавита невозможно представить в таком виде, не добавив новых связей между ними, KNN дает такой высокий результат.

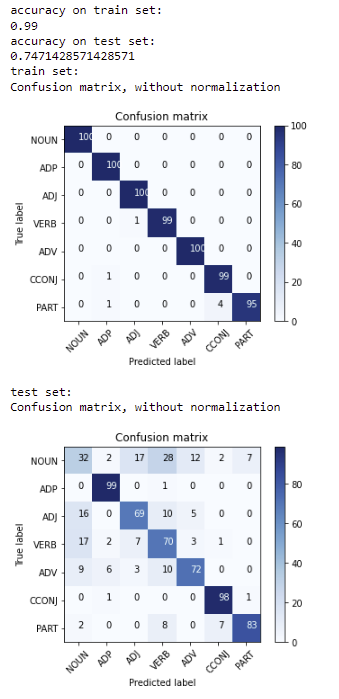
Мое предположение заключается в том, что, благодаря обработке входных слов, алгоритм часто находит среди данных для обучения слово, массив признаков которого после обработки в точности соответствует массиву признаков искомого слова. То, что соседние буквы алфавита дают похожий результат, не должно негативно влиять на точность классификации таких примеров, а ошибки должны чаще возникать, когда отличие находится в коде буквы, а не в длине.

Данную гипотезу можно проверить двумя способами.

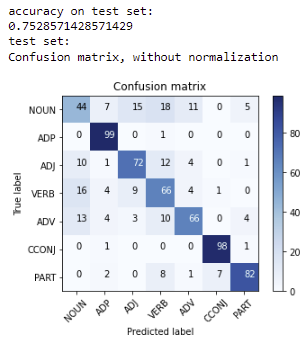
1. Увеличить расстояние между соседними кодами букв, чтобы расстояние между объектами с соседними буквами алфавита было существенно больше, чем расстояние между объектами с близкими длинами. Тогда алгоритм будет, где это возможно, выбирать пары слов, содержащие одни и те же буквы на ключевых позициях, а не пары слов одинаковой длины. Точность должна увеличиться. Точно также, если увеличить расстояние между соседними длинами, точность должна упасть.
2. Проверить, какие различия приводят к ошибкам, напрямую посмотрев данные о ближайшем соседе.

Так как наибольшую точность дает значение k = 1, т.е. алгоритм ищет лишь одного ближайшего соседа, будем экспериментировать, используя его.

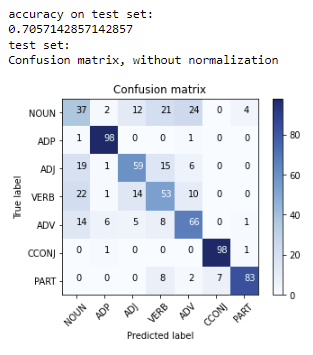
Расстояние между соседними значениями длины равно расстоянию между буквами



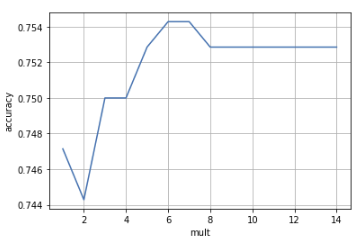
Расстояние между буквами больше, чем между длинами в 10 раз



Расстояние между длинами больше, чем между буквами в 10 раз



Как видно, первая гипотеза подтвердилась. По матрице спутывания видно, что при увеличении расстояния между буквами, точность определения некоторых классов выросла, в то время, как других – упала. Возможно, можно подобрать такой множитель, что результирующая общая точность будет еще больше.

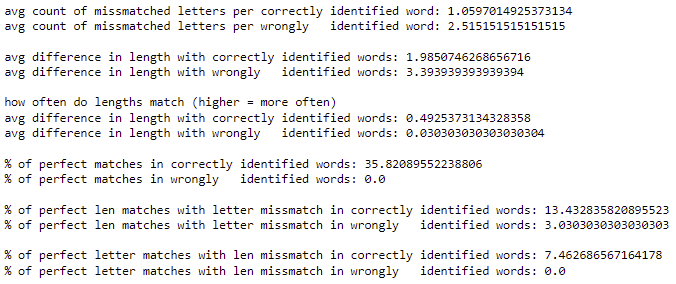


Наиболее вымокая точность, которую удалось достичь:



Она не сильно больше, чем точность при равном расстоянии между длинами и буквами.

Для проверки второго предположения, я, используя KNN с числом соседей k = 1, выводил элемент, который оказался наиболее близким к классифицируемому, и сравнивал их между собой. Сравнения дали такой результат:



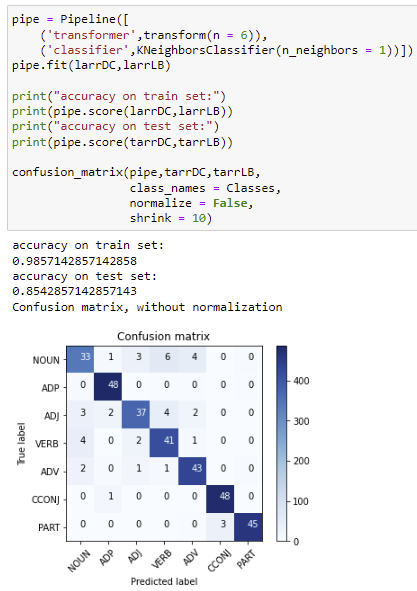
В целом, ничего удивительного. Чем дальше слово от ближайшего соседа, тем скорее оно будет определено неверно.

Интересно, что среди слов, классифицированных неверно, есть слова, у которых совпала длина, но не совпали буквы, но нет слов, у которых совпали буквы, но не совпала длина, т.е. все такие слова были определены верно. Это подтверждает то, что для верного определения части речи с помощью KNN важнее, чтобы совпали буквы на ключевых позициях, а не длина. Также всегда, когда признаки слова совпадали с ближайшим соседом, его часть речи определялась верно.

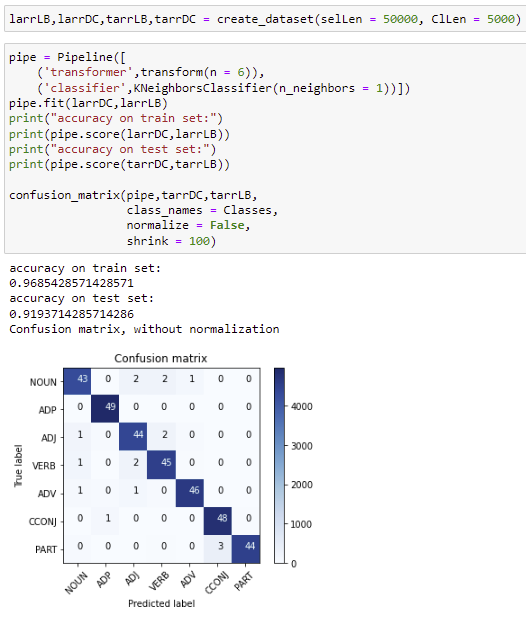
Разумеется, стоит помнить, что это исследование было проведено на выборке в 700 слов, и потому, точные цифры могут несколько отличаться, однако, как вытаскивать ближайших соседей из классификатора из skl я не знаю, а на моем классификаторе большее количество слов классифицировать будет слишком долго.

В отличие от моей реализации, KNN из модуля skl работает очень быстро и не сильно замедляется при увеличении выборок.

Пример классификатора KNN, созданного с помощью модуля skl. И классифицирующего сбалансированный датасет в 3500 слов (числа на матрице спутывания уменьшены в 10 раз)

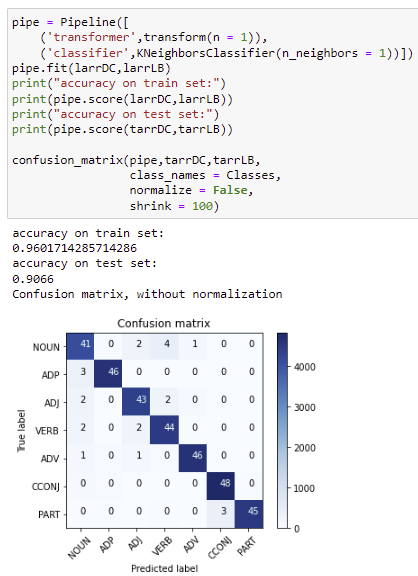


Использование KNN на датасете длинною в 7\*5000 (35000) слов (числа на матрице спутывания уменьшены в 100 раз)

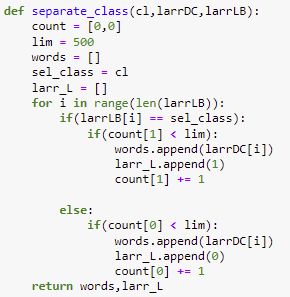


Что неудивительно, с увеличением обучающей выборки точность на тестовом датасете растет.

Также, в этом примере гораздо более выражено то, что умножение кодов букв действительно увеличивает точность. Та же модель, но с мультипликатором трансформера = 1:

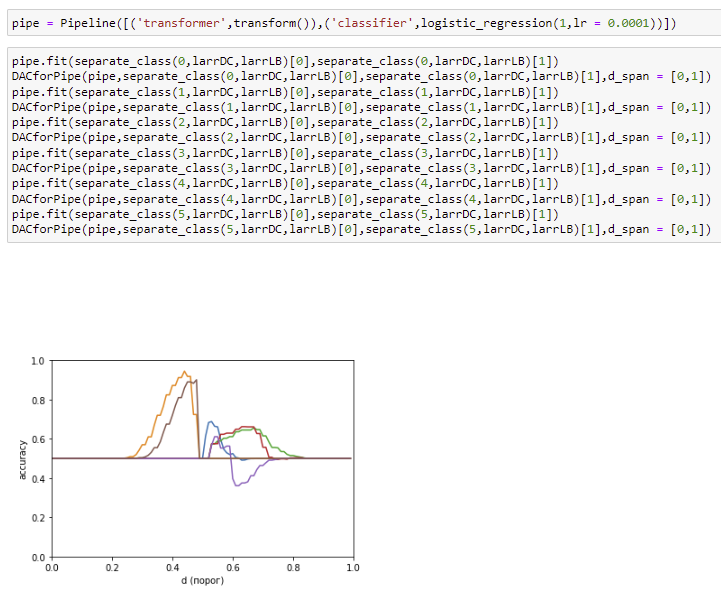


Раз уж KNN оказалась неожиданно эффективной в классификации слов по частям речи, стоит также проверить, как покажут себя модели-регрессии. Очевидно, что для реализации n-class classification с помощью регрессий, предназначенных для бинарной классификации, нужно решать задачу выделения класса k. Для этого напишем такой модификатор датасета:



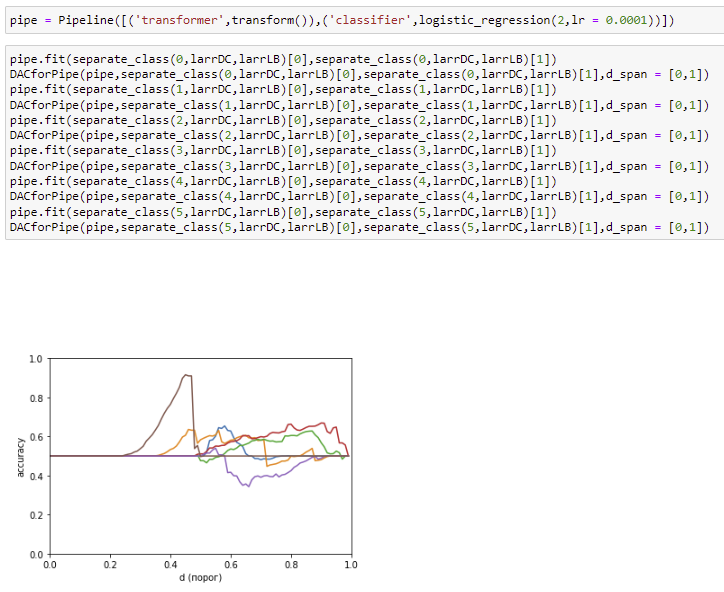
Он преобразует датасет с n классами в сбалансированный бинарный датасет, где класс cl обозначен единицей, а остальные классы – нулем. Так как эта функция воздействует не только на значения признаков, но и на лейблы, представить ее в виде трансформера в конвейере не получилось.

Попробуем выделить каждый класс, имея в качестве признаков только длину слова:

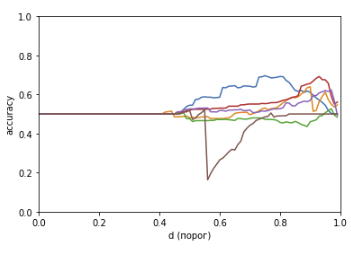


Единственные два класса, которые получилось выделить хорошо (высокие пики на графике) – это предлог и союз. Это было ожидаемо, однако, с введением других признаков, ситуация становится только хуже.

**Длина + последняя буква**



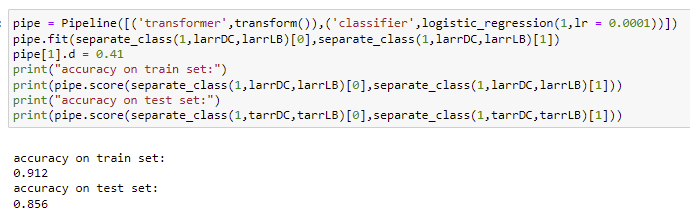
**Все признаки**



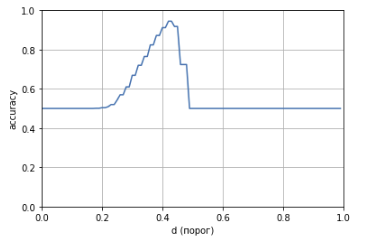
(я мог бы подписать графики, но и так видно, что ничего не работает, так что какая разница)

Разумеется, регрессии можно применять для весьма точного выделения предлогов по длине, но ту же задачу можно решить, просто вручную задав порог длины в 3-4 буквы, которые предлоги редко преодолевают. Тот же KNN прекрасно справляется с задачей выделения предлогов внутри задачи многоклассовой классификации, что видно по его матрице спутывания.

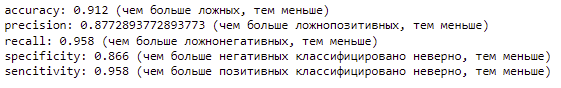
**Пример реализации задачи выделения предлогов с помощью логистической регрессии**

****

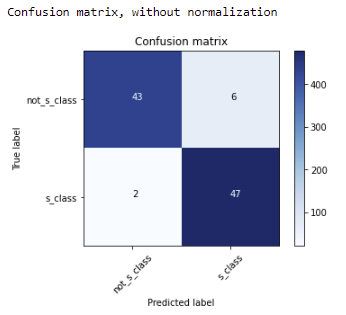
Кривая порог-точность

****

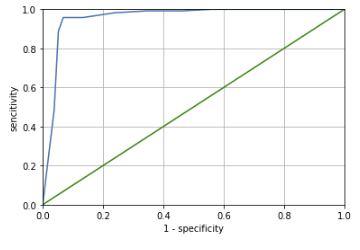
Базовые метрики

****

Матрица спутывания

****

Кривая ROC

****

2.3. Сохранение наиболее эффективной модели.

Очевидно, что наиболее эффективной для решения данной задачи моделью оказалась KNN. Она единственная показала приемлемый результат и, на большом датасете перешла границу точности в 90% на тестовой выборке. На втором месте оказался наивный байесовский классификатор с точностью ~60%, остальные модели не удалось эффективно применить для решения поставленной задачи. Сохраним эти две модели.



1. **Выводы**

В ходе выполнения этой лабораторной работы я реализовал классификаторы LR, LogR, SVM, KNN и NB, определил наиболее подходящий классификатор для решения задачи классификации слов русского языка по частям речи и подобрал оптимальные гиперпараметры.

Я научился использовать конвейеры в skl и стал лучше понимать принципы работы и основные отличия вышеперечисленных алгоритмов. Также я получил опыт работы с настоящим nlp-датасетом, что может пригодиться мне в будущем.

У меня получилось добиться классификации тестового датасета с точностью в 91.94%, что больше, чем я ожидал, но все же недостаточно хорошо, чтобы использовать такой классификатор в настоящих nlp-системах. Возможно, точность можно улучшить, совместив несколько разнотипных классификаторов (KNN + NB), либо используя бутстрап или бустинг.

Интересно, что KNN показал наибольшую точность как при испытании на искусственном датасете, так и при решении основной задачи. Единственным, но при этом критическим недостатком этого алгоритма оказалась низкая скорость классификации, хотя реализация в skl работает относительно быстро.